The Impact of Active Domain Predicates on Guarded Existential Rules

Fundamenta Informaticae XX (2017) 1–22
DOI 10.3233/FI-2015-0000
IOS Press

The Impact of Active Domain Predicates on Guarded Existential Rules

Georg Gottlob
Department of Computer Science
University of Oxford
Oxford, UK
georg.gottlob@cs.ox.ac.uk

Andreas Pieris
School of Informatics
University of Edinburgh
Edinburgh, UK
apieris@inf.ed.ac.uk

Mantas Šimkus
Institute of Information Systems
Vienna University of Technology
Vienna, Austria
simkus@dbai.tuwien.ac.at

Abstract. It is realistic to assume that a database management system provides access to the active domain via built-in relations. Therefore, databases that include designated predicates that hold the active domain, which we call product databases, form a natural notion that deserves our attention. An important issue then is to look at the consequences of product databases for the expressiveness and complexity of central existential rule languages. We focus on guarded-based existential rules, and we investigate the impact of product databases on their expressive power and complexity. We show that the queries expressed via (frontier-)guarded rules gain in expressiveness, and in fact, they have the same expressive power as Datalog. On the other hand, there is no impact on the expressiveness of the queries specified via weakly-(frontier-)guarded rules since they are powerful enough to explicitly compute the predicates needed to access the active domain. We also observe that there is no impact on the complexity of the query languages in question.

Keywords: Rule-based languages, ontology-mediated queries, conjunctive queries, existential rules, tuple-generating dependencies, guardedness, expressive power, complexity

1. Introduction

Rule-based languages lie at the core of databases and knowledge representation. In database applications they are usually employed as query languages that go beyond standard SQL, while in knowledge representation are used for declarative problem solving, and, more recently, to model and reason about ontological knowledge. Therefore, rule-based languages can be used in at least two different ways: as query languages and as ontology languages.

In the database setting, a rule-based query is expressed as a pair of the form $(\Sigma, \texttt{Ans})$, where Σ is a set of rules encoding the actual query, while $\texttt{Ans}$ is the so-called goal predicate that collects the answer to the query. For example, if we want to compute the transitive closure of a graph $G = (V, E)$, which is encoded in a database in the usual way, then Σ consists of

$$E(x, y) \rightarrow T(x, y) \qquad E(x, y), T(y, z) \rightarrow T(x, z) \qquad T(x, y) \rightarrow \texttt{Ans}(x, y),$$

where the first rule copies all the edges of G into the binary predicate T, the second rule recursively computes the transitive closure of G, while the third rule, which is the output rule, simply copies the binary predicate T into the output predicate $\texttt{Ans}$.

On the other hand, in the ontological setting, a database D and a set of rules Σ are used to specify implicit domain knowledge – the pair (D, Σ) is called *knowledge base* – while user queries, typically expressed as standard conjunctive queries, are evaluated over a knowledge base. For example, the knowledge base (D, Σ), where $D = \{\texttt{Person}(\text{“Alice”})\}$ and Σ consists of the rules

$$\texttt{Person}(x) \rightarrow \exists y\, \texttt{HasMother}(x, y) \qquad \texttt{HasMother}(x, y) \rightarrow \texttt{Person}(y),$$

states that Alice is a person, while each person has a mother who is also a person. Now, the query whether Alice has a mother who is also a person can be easily expressed using the conjunctive query

$$q = \exists x(\texttt{HasMother}(\text{“Alice”}, x) \wedge \texttt{Person}(x)),$$

which is clearly entailed by (D, Σ). Alternatively, the set of rules Σ and the conjunctive query q can be conceived as the components of one composite query, called *ontology-mediated query*, which will be then evaluated over the database D. So, ontology-mediated queries are in fact pairs of the form (Σ, q), where Σ is a set of rules expressed in a certain ontology language, and q is a conjunctive query [1].[1]

From the above discussion, it is apparent that rule-based languages form the building block of several database and ontology-mediated query languages that can be found in the literature. An important issue for a query language (either a database or an ontology-mediated query language) is to understand its expressive power, and in particular, its expressiveness relative to other query languages. *Relative expressiveness* considers if, given two query languages L_1 and L_2, every query formulated in L_1 can be expressed by means of L_2 (and vice versa). This helps the user to choose, among a plethora of different

[1] In general, ontology-mediated queries are defined for arbitrary ontology languages, which can be seen as fragments of first-order logic, and arbitrary database query languages. However, in this work, we focus on rule-based ontology languages, and database queries definable via conjunctive queries.

query languages, the one that is more appropriate for the application in question. The goal of this work is to perform such an expressivity analysis for central query languages based on existential rules.

Existential rules (a.k.a. *tuple-generating dependencies* or *Datalog$^{\pm}$ rules*) are first-order sentences of the form $\forall\bar{x}\forall\bar{y}\big(\phi(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi(\bar{x},\bar{z})\big)$, where ϕ and ψ are conjunctions of atoms – usually, for brevity, we omit the universal quantification, and use comma instead of "$\wedge$" for conjoining atoms. Intuitively, such a rule states that the existence of certain tuples in a database implies the existence of some other tuples in the same database. It is widely known that the query languages based on arbitrary existential rules, without posing any syntactic restriction, are undecidable; see, e.g., [2, 3]. This has led to a flurry of activity for identifying expressive fragments of existential rules that give rise to decidable query languages. One of the key paradigms that has been thoroughly studied is guardedness [3, 4]. In a nutshell, the existential rule given above is guarded (resp., frontier-guarded) if ϕ has an atom that contains (or "guards") all the variables in $\bar{x} \cup \bar{y}$ (resp., $\bar{x}$). More refined languages based on weak-(frontier-)guardedness also exist. All these languages form the so-called guarded family of existential rules.

The relative expressiveness of the languages in the guarded family of existential rules has been recently investigated in [5]. However, the thorough analysis performed in [5] has made no assumption on the input databases over which the queries will be evaluated. We claim it is realistic to assume that a database management system gives us free access to the active domain, that is, the set of constant values occurring in the underlying database, via built-in relations (e.g., lookup or reference tables). In other words, we assume that a database system has a mechanism for checking whether a given tuple of values consists only of constants in the active domain. In more formal terms, this means that the underlying database is what we call a *product database*, that is, a database that includes designated predicates that hold the active domain; in fact, those predicates gives us access to the cartesian product of the active domain, and hence the name "product database". The question that comes up is whether the relative expressiveness of the languages in the guarded family of existential rules is affected when we concentrate on product databases. We show that:

- The query languages based on (frontier-)guarded existential rules gain in expressiveness, and, in fact, they have the same expressive power as Datalog.
- There is no impact on the expressive power of the query languages that are based on weakly-(frontier-)guarded existential rules, since they are powerful enough to explicitly compute the relations needed to access the active domain.

At this point, since the query languages based on (frontier-)guarded existential rules over product databases have the same expressive power as plain Datalog, one may ask why those languages are useful or relevant. Recall that a Datalog query is a query of the form $(\Sigma, \texttt{Ans})$, where Σ is a set of rules of the form $\phi(\bar{x},\bar{y}) \rightarrow R(\bar{x})$, i.e., existential rules with a single-atom existential-free head, while $\texttt{Ans}$ is the output predicate. Therefore, the above question simply asks the following: what is the advantage of allowing existentially quantified variables to appear in the head of a rule? This question has already been addressed in the context of querying RDF data, where a refined notion of expressive power, called *program expressive power*, has been introduced [6]. The key idea underlying this new notion of expressive power is to encode the set of conjunctive queries that can be made true over all databases via a ***fixed*** set of rules (a.k.a. program, hence the name "program expressive power"). We show that:

- Assuming product databases, the query languages based on (frontier-)guarded existential rules are strictly more expressive than Datalog w.r.t. the program expressive power.

We also consider the complexity of the query languages in question, and we observe that:

- Even if we focus on product databases, the existing query evaluation algorithms can be applied, and we get the same complexity results as in the case of arbitrary (non-product) databases.

Finally, we discuss how the above results can be used in order to define novel classes of existential rules that give rise to query languages that are equally expressive to Datalog, data tractable, and closed under union. The latter is a crucial property that enables a modular style of writing queries.

Although the employed techniques for establishing our results are rather standard, which build on existing ones that can be found in the literature, the obtained results are conceptually interesting (e.g., assuming product databases, (frontier-)guardedness gives rise to query languages that are equally expressive to Datalog). Our analysis sheds light on the expressivity of the guarded-based query languages in question, and complements the recent investigation performed in [5]. In the above summarization of our results, the term query language refers to both database and ontology-mediated query languages. Since the former is a special case of the latter[2], in the sequel we focus on ontology-mediated queries.

2. A First Glimpse on Product Databases

The goal of this section is to illustrate, via a meaningful example, that product databases have an impact on the expressiveness of frontier-guarded ontology-mediated queries, which in turn allows us to write complex queries in a more convenient and flexible way. Suppose we are developing a system for managing a response to a natural disaster. The ultimate goal of the system is to collect information about volunteers and their qualifications, and then use this information to coordinative various relief activities.

The Database

Suppose that the database of such a system contains a binary relation `Team` that stores an assignment of volunteers to teams. For example, the atom

$$\texttt{Team}(\text{“Alpha”}, \text{“Ann”})$$

means that Ann belongs to the team called Alpha. The database also includes a binary relation called `ExperienceIn`, which relates persons to tasks in which they have experience. For instance, the atom

$$\texttt{ExperienceIn}(\text{“John”}, \text{“perform CPR”})$$

states that John has experience in performing CPR. We also have a binary relation `hasTraining` with the obvious meaning; for example,

$$\texttt{hasTraining}(\text{“John”}, \text{“race driver”})$$

means that John has been trained to drive a race car. In addition, the database contains a unary relation `ProDriverQualification` that stores qualifications that involve driving at professional level; e.g.,

$$\texttt{ProDriverQualification}(\text{“bus license”})$$

[2] Indeed, the query $(\Sigma, \texttt{Ans})$ is actually the ontology-mediated query $(\Sigma, \texttt{Ans}(x_1, \ldots, x_n))$, where n is the arity of `Ans`.

states that bus license is a qualification to drive at professional level. We further assume that some tasks that can be performed by volunteers are grouped into more complex procedures. For instance, the response to a water leak could consist of performing four tasks in the following order: load equipment, drive truck, perform repairs and clean up. This is stated in the database of the system using the atoms:

$$\texttt{ProcedureTaskFirst}(\text{“water leak”}, \text{“load equipment”})$$
$$\texttt{ProcedureTaskOrder}(\text{“water leak”}, \text{“load equipment”}, \text{“drive truck”})$$
$$\texttt{ProcedureTaskOrder}(\text{“water leak”}, \text{“drive truck”}, \text{“perform repairs”})$$
$$\texttt{ProcedureTaskOrder}(\text{“water leak”}, \text{“perform repairs”}, \text{“clean up”})$$
$$\texttt{ProcedureTaskLast}(\text{“water leak”}, \text{“clean up”}).$$

Intuitively, $\texttt{ProcedureTaskFirst}(p, t)$ and $\texttt{ProcedureTaskLast}(p, t')$ state that t and t' are the first and the last tasks to be performed in the procedure p. An atom of the form $\texttt{ProcedureTaskOrder}(p, t, t')$ means that in the procedure p the task t' follows the task t.

The Ontology

We know that some intensional knowledge, not explicitly stored in the database described above, also holds. More precisely, we know that if a person p has experience in some task t, then p is qualified to perform t. This can be formally expressed via the rule

$$\sigma_1 \;=\; \texttt{ExperienceIn}(Pn, Tk) \;\rightarrow\; \texttt{QualifiedFor}(Pn, Tk).$$

Moreover, we know that if a person p has been trained to be a professional driver, then p is qualified to drive an ambulance. This can be expressed as

$$\sigma_2 \;=\; \texttt{hasTraining}(Pn, T), \texttt{ProDriverQualification}(T) \;\rightarrow\; \texttt{QualifiedFor}(Pn, \text{“drive ambulance”}).$$

In addition, if a person p is experienced in delivering heavy goods, then p must have some training that leads to a truck license. This is expressed via the rule

$$\sigma_3 \;=\; \texttt{ExperienceIn}(Pn, \text{“delivery heavy goods”}) \;\rightarrow\; \exists T\, \texttt{hasTraining}(Pn, T), \texttt{TruckLicense}(T).$$

Finally, truck license leads to a professional driving license, which can be expressed as

$$\sigma_4 \;=\; \texttt{TruckLicense}(T) \;\rightarrow\; \texttt{ProDriverQualification}(T).$$

Observe that our ontology $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$ consists of guarded existential rules.

The Database Query

In our disaster management scenario we are interested in checking whether a team is qualified to perform every task of a certain procedure. More precisely, we want to collect in a binary relation `TeamQualified`

all pairs (t, p) of a team and a procedure such that: for every task j of the procedure p, the team t has a member m that is qualified for j. Recall that an ontology-mediated query is a pair of an ontology and a database query. Therefore, we need to express the above query as a database query q, which, together with the ontology Σ defined above, will give rise to the ontology-mediated query (Σ, q). Unfortunately, things are a bit more complicated than they seem. In particular, the query q is inherently recursive, and thus is not expressible as a conjunctive query. However, it can be easily expressed as the Datalog query $(\Pi, \texttt{TeamQualified})$, where the program Π consists of the rules:

$$\begin{aligned} & \texttt{ProcedureTaskFirst}(Pc, Tk), \\ \rho_1 \;=\; & \texttt{Team}(Tm, Pn), \\ & \texttt{QualifiedFor}(Pn, Tk) \;\rightarrow\; \texttt{QualifiedUntil}(Tm, Pc, Tk) \end{aligned}$$

$$\begin{aligned} & \texttt{ProcedureTaskOrder}(Pc, Tk', Tk), \\ \rho_2 \;=\; & \texttt{Team}(Tm, Pn), \\ & \texttt{QualifiedUntil}(Tm, Pn, Tk') \;\rightarrow\; \texttt{QualifiedUntil}(Tm, Pc, Tk) \end{aligned}$$

$$\begin{aligned} \rho_3 \;=\; & \texttt{ProcedureTaskLast}(Pc, Tk), \\ & \texttt{QualifiedUntil}(Tm, Pc, Tk) \;\rightarrow\; \texttt{TeamQualified}(Tm, Pc). \end{aligned}$$

The fact that our query q is expressible as a recursive Datalog query is of little use since the ontology-mediated query (Σ, q) does not comply with the formal definition of ontology-mediated queries where q must be a first-order query, and thus does not fall in a decidable guarded-based ontology-mediated query language. Hence, the crucial question that comes up is whether we can construct a query (Σ', q') that is equivalent to (Σ, q), while Σ is a set of (frontier-)guarded existential rules and q is a conjunctive query. One may think that this can be achieved by adding the Datalog rules of Π in the ontology Σ, i.e., $\Sigma' = \Sigma \cup \Pi$, and let q be the atomic conjunctive query $\texttt{TeamQualified}(x, y)$. Although the obtained query (Σ', q') is equivalent to (Σ, q), it is inherently unguarded, and it can be shown that cannot be expressed as a frontier-guarded ontology-mediated query. However, by adopting the natural assumption that our database is a product database, which gives us access to the active domain via relations of the form $\texttt{Dom}^k$, for $k > 0$, that hold all the k-tuples of constants occurring in the active domain, we can replace the rules $\rho_1, \rho_2 \in \Sigma'$ with the guarded rules

$$\begin{aligned} & \texttt{ProcedureTaskFirst}(Pc, Tk), \\ \rho_1' \;=\; & \texttt{Team}(Tm, Pn), \\ & \texttt{QualifiedFor}(Pn, Tk) \\ & \texttt{Dom}^4(Tm, Pc, Tk, Pn) \;\rightarrow\; \texttt{QualifiedUntil}(Tm, Pc, Tk) \end{aligned}$$

$$\begin{aligned} & \texttt{ProcedureTaskOrder}(Pc, Tk', Tk), \\ \rho_2' \;=\; & \texttt{Team}(Tm, Pn), \\ & \texttt{QualifiedUntil}(Tm, Pn, Tk'), \\ & \texttt{Dom}^5(Tm, Pc, Tk, Tk', Pn) \;\rightarrow\; \texttt{QualifiedUntil}(Tm, Pc, Tk) \end{aligned}$$

without changing the meaning of the query (Σ', q'). Hence, the assumption that the database is a product database allows us to rewrite the query (Σ, q) into an equivalent guarded ontology-mediated query.

3. Ontology Mediated-Query Languages

Instances and Queries. Let $\mathbf{C}$, $\mathbf{N}$ and $\mathbf{V}$ be pairwise disjoint countably infinite sets of *constants*, (labeled) *nulls* and *variables* (used in queries and dependencies), respectively. A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity n. A *term* is either a constant, null or variable. An *atom* over $\mathbf{S}$ is an expression $R(\bar{t})$, where R is a relation symbol in $\mathbf{S}$ of arity $n > 0$ and $\bar{t}$ is an n-tuple of terms. A *fact* is an atom whose arguments consist only of constants. An *instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that contain constants and nulls, while a *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of an instance I, denoted $adom(I)$, is the set of all terms occurring in I.

A *query* over $\mathbf{S}$ is a mapping q that maps every database D over $\mathbf{S}$ to a set of *answers* $q(D) \subseteq adom(D)^n$, where $n \geq 0$ is the *arity* of q. The usual way of specifying queries is by means of (fragments of) first-order logic. Such a central fragment is the class of conjunctive queries. A *conjunctive query* (CQ) q over $\mathbf{S}$ is a conjunction of atoms of the form $\exists \bar{y}\, \phi(\bar{x}, \bar{y})$, where $\bar{x} \cup \bar{y}$ are variables of $\mathbf{V}$, that uses only predicates from $\mathbf{S}$. The free variables of a CQ are called *answer variables*. The evaluation of CQs over instances is defined in terms of homomorphisms. A *homomorphism* from a set of atoms A to a set of atoms A' is a partial function $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ such that: (i) $t \in \mathbf{C}$ implies $h(t) = t$, i.e., is the identity on $\mathbf{C}$, and (ii) $R(t_1, \ldots, t_n) \in A$ implies $h(R(t_1, \ldots, t_n)) = R(h(t_1), \ldots, h(t_n)) \in A'$. The *evaluation* of q over an $\mathbf{S}$-instance I, denoted $q(I)$, is the set of all tuples $h(\bar{x})$ of constants such that h is a homomorphism from q to I. Each schema $\mathbf{S}$ and CQ $q = \exists \bar{y}\, \phi(x_1, \ldots, x_n, \bar{y})$ give rise to the n-ary query $q_{\phi,\mathbf{S}}$ defined by setting, for every database D over $\mathbf{S}$, $q_{\phi,\mathbf{S}}(D) = \{\bar{c} \in adom(D)^n \mid \bar{c} \in q(D)\}$. Let CQ be the class of all queries definable by some CQ.

Tgds for Specifying Ontologies. An ontology language is a fragment of first-order logic. In this work, we focus on ontology languages that are based on tuple-generating dependencies. A *tuple-generating dependency* (tgd) is a first-order sentence of the form

$$\forall \bar{x} \forall \bar{y} \big(\phi(\bar{x}, \bar{y}) \;\rightarrow\; \exists \bar{z}\, \psi(\bar{x}, \bar{z}) \big),$$

where both ϕ and ψ are conjunctions of atoms without nulls and constants. For simplicity, we write this tgd as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$, and use comma instead of "$\wedge$" for conjoining atoms. We call ϕ and ψ the *body* and *head* of the tgd, respectively; for a tgd σ, we write $body(\sigma)$ and $head(\sigma)$ for the body and the head of σ, respectively. Let $sch(\Sigma)$ be the set of predicates occurring in a set of tgds Σ. An instance I *satisfies* the above tgd if: For every homomorphism h from $\phi(\bar{x}, \bar{y})$ to I, there is a homomorphism h' that extends h, i.e., $h' \supseteq h$, from $\psi(\bar{x}, \bar{z})$ to I. I satisfies a set Σ of tgds, denoted $I \models \Sigma$, if I satisfies every tgd in Σ. Let TGD be the class of all (finite) sets of tgds.

Ontology-Mediated Queries. An *ontology-mediated query* is a triple $(\mathbf{S}, \Sigma, q)$, where $\mathbf{S}$ is a schema, called *data schema*, $\Sigma \in \mathsf{TGD}$, $q \in \mathsf{CQ}$, and q is over $\mathbf{S} \cup sch(\Sigma)$.[3] Notice that the data schema $\mathbf{S}$ is included in the specification of an ontology-mediated query in order to make clear that the query is over $\mathbf{S}$, i.e., it ranges over $\mathbf{S}$-databases. The semantics of such a query is defined in terms of certain answers.

[3] In fact, ontology-mediated queries can be defined for arbitrary ontology and query languages.

Let $(\mathbf{S}, \Sigma, q)$ be an ontology-mediated query, where n is the arity of q. The *answer* to q with respect to a database D over $\mathbf{S}$ and Σ is the set of tuples

$$cert_{q,\Sigma}(D) \;=\; \bigcap_{I \supseteq D, I \models \Sigma} \{\bar{c} \in adom(D)^n \mid \bar{c} \in q(I)\}.$$

At this point, it is important to recall that $cert_{q,\Sigma}(D)$ coincides with the evaluation of q over the canonical instance of D and Σ that can be constructed by applying the chase procedure [3, 7, 8, 9]. Roughly speaking, the chase adds new atoms to D as dictated by Σ until the final result satisfies Σ, while the existentially quantified variables are satisfied by inventing fresh null values. The formal definition of the chase procedure follows. Let I be an instance and $\sigma = \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z})$ a tgd. We say that σ is *applicable* with respect to I if there exists a homomorphism h from $body(\sigma)$ to I. In this case, *the result of applying σ over I with h* is the instance $J = I \cup h'(head(\sigma))$, where h' is an extension of h that maps each variable $z \in \bar{z}$ to a fresh null value not occurring in I. For such a single chase step we write $I \xrightarrow{\sigma,h} J$. Let us assume now that I is an instance and Σ a finite set of tgds. A *chase sequence for I under Σ* is a (finite or infinite) sequence:

$$I_0 \xrightarrow{\sigma_0,h_0} I_1 \xrightarrow{\sigma_1,h_1} I_2 \ldots$$

of chase steps such that: (1) $I_0 = I$; (2) For each $i \geq 0$, $\sigma_i \in \Sigma$; and (3) $\bigcup_{i \geq 0} I_i \models \Sigma$. Notice that in case the above chase sequence is infinite, then it must be also *fair*, that is, whenever a tgd $\sigma \in \Sigma$ is applicable with respect to I_i with homomorphism h_i, then there exists $h' \supseteq h_i$ and $k > i$ such that $h'(head(\sigma)) \subseteq I_k$. In other words, a fair chase sequence guarantees that all tgds that are applicable will eventually be applied. We call $\bigcup_{i \geq 0} I_i$ the *result* of this chase sequence, which always exists. Although the result of a chase sequence is not necessarily unique (up to isomorphism), each such result is equally useful for our purposes since is *universal*, that is, it can be homomorphically embedded into every other result. Therefore, we denote by $chase(I, \Sigma)$ *the* result of an arbitrary chase sequence for I under Σ.

Given an ontology-mediated query $(\mathbf{S}, \Sigma, q)$, it is well-known that $cert_{q,\Sigma}(D) = q(chase(D, \Sigma))$, for every $\mathbf{S}$-database D. In other words, to compute the answer to q with respect to D and Σ, we simply need to evaluate q over the instance $chase(D, \Sigma)$. Notice that this does not provide an effective algorithm for computing $cert_{q,\Sigma}(D)$ since the instance $chase(D, \Sigma)$ is, in general, infinite.

Ontology-Mediated Query Languages. Every ontology-mediated query $Q = (\mathbf{S}, \Sigma, q)$ can be interpreted as a query q_Q over $\mathbf{S}$ by setting $q_Q(D) = cert_{q,\Sigma}(D)$, for every $\mathbf{S}$-database D. Thus, we obtain a new query language, denoted $(\mathsf{TGD}, \mathsf{CQ})$, defined as the class of queries q_Q, where Q is an ontology-mediated query. However, $(\mathsf{TGD}, \mathsf{CQ})$ is undecidable since, given a database D over $\mathbf{S}$, $\Sigma \in \mathsf{TGD}$, an n-ary query $q \in \mathsf{CQ}$ over $\mathbf{S} \cup sch(\Sigma)$, and a tuple $\bar{c} \in \mathbf{C}^n$, the problem of deciding whether $\bar{c} \in cert_{q,\Sigma}(D)$ is undecidable; see, e.g., [2, 3]. This has led to a flurry of activity for identifying decidable syntactic restrictions. Such a restriction defines a subclass $\mathcal{C}$ of tgds, i.e., $\mathcal{C} \subseteq \mathsf{TGD}$, which in turn gives rise to the query language $(\mathcal{C}, \mathsf{CQ})$. Such a query language is called *ontology-mediated query language*. Here we focus on ontology-mediated query languages that are based on the notion of guardedness:

(Frontier-)Guarded Tgds: A tgd is *guarded* if its body contains an atom, called *guard*, that contains all the body-variables [3]. Let G be the class of all finite sets of guarded tgds. A key extension of guarded tgds is the class of *frontier-guarded* tgds, where the guard contains only the frontier variables, i.e., the body-variables that appear in the head [4]. Let FG be the class of all finite sets of frontier-guarded tgds.

Weak Versions: Both G and FG have a weak version: Weakly-guarded [3] and weakly-frontier-guarded [4], respectively. These are highly expressive classes of tgds obtained by relaxing the underlying condition so that only those variables that may unify with null values during the chase are taken into account. In order to formalize these classes of tgds we need some additional terminology.

A *position* $R[i]$ identifies the i-th attribute of a predicate R. Given a schema $\mathbf{S}$, the set of positions of $\mathbf{S}$ is the set $\{R[i] \mid R/n \in \mathbf{S} \text{ and } i \in \{1, \ldots, n\}\}$. Given a set Σ of tgds, the set of *affected positions* of $sch(\Sigma)$, denoted $affected(\Sigma)$, is inductively defined as follows: (1) If there exists $\sigma \in \Sigma$ such that at position π an existentially quantified variable occurs, then $\pi \in affected(\Sigma)$; and (2) If there exists $\sigma \in \Sigma$ and a variable V in $body(\sigma)$ only at positions of $affected(\Sigma)$, and V appears in $head(\sigma)$ at position π, then $\pi \in affected(\Sigma)$. A tgd σ is *weakly-guarded with respect to* Σ if its body contains an atom, called *weak-guard*, that contains all the body-variables that appear only at positions of $affected(\Sigma)$. The set Σ is *weakly-guarded* if each $\sigma \in \Sigma$ is weakly-guarded with respect to Σ. The class of weakly-frontier-guarded sets of tgds is defined analogously, but considering only the body-variables that appear also in the head of a tgd. We write WG (resp., WFG) for the class of all finite weakly-guarded (resp., weakly-frontier-guarded) sets of tgds.

4. Product Databases

Recall that product databases provide access to the active domain via designated built-in predicates. Before proceeding to the next section, where we look at the impact of product databases on the expressive power of the ontology-mediated query languages in question, let us make the notion of product databases more precise.

A database D is said to be *α-product*, where α is a finite set of positive integers, if it includes a designated predicate $\mathtt{Dom}^i/i$, for each $i \in \alpha$, that holds all the i-tuples of constants in $adom(D)$, or, in other words, the restriction of D over the predicate $\mathtt{Dom}^i$ is precisely the set of facts $\{\mathtt{Dom}^i(\bar{t}) \mid \bar{t} \in adom(D)^i\}$. Given a non-product database D, we denote by D^α the α-product database $D \cup \{\mathtt{Dom}^i(\bar{t}) \mid \bar{t} \in adom(D)^i\}_{i \in \alpha}$. An *ontology-mediated query over a product database* is an ontology-mediated query $(\mathbf{S}, \Sigma, q)$ such that $\mathbf{S}$ contains the predicates $\mathtt{Dom}^{i_1}, \ldots, \mathtt{Dom}^{i_k}$, for some set of positive integers $\alpha = \{i_1, \ldots, i_k\}$, while none of those predicates appears in the head of a tgd of Σ. The latter condition is posed since the predicates $\mathtt{Dom}^{i_1}, \ldots, \mathtt{Dom}^{i_k}$ are conceived as built-in read-only predicates, and thus, we cannot modify their content. Such a query ranges only over $\mathbf{S}$-databases that are α-product. We write $(\mathcal{C}, \mathsf{CQ})_\times$ for the class of $(\mathcal{C}, \mathsf{CQ})$ queries over a product database.

Example 4.1. Consider the query $Q_{trans} = (\{E\}, \Sigma, \mathtt{Ans}(x, y))$, where Σ is the set:

$$\begin{aligned} E(x, y) &\rightarrow T(x, y) \\ E(x, y), T(y, z) &\rightarrow T(x, z) \\ T(x, y) &\rightarrow \mathtt{Ans}(x, y), \end{aligned}$$

which computes the transitive closure of the binary predicate E. It is easy to see that the above query can be equivalently rewritten as a guarded ontology-mediated query over a product database, i.e., as a $(\mathsf{G}, \mathsf{CQ})_\times$ query. More precisely, Q_{trans} can be written as $Q'_{trans} = (\{E, \mathtt{Dom}^3\}, \Sigma', \mathtt{Ans}(x, y))$, where

Σ' is the set of tgds:

$$\begin{aligned} E(x,y) &\rightarrow T(x,y) \\ \texttt{Dom}^3(x,y,z), E(x,y), T(y,z) &\rightarrow T(x,z) \\ T(x,y) &\rightarrow \texttt{Ans}(x,y). \end{aligned}$$

Clearly, for every $\{E\}$-database D, $Q_{trans}(D) = Q'_{trans}(D^{\{3\}})$. ∎

5. The Impact of Product Databases

We are now ready to investigate the impact of product databases on the relative expressiveness of the guarded-based ontology-mediated query languages in question. But let us first fix some auxiliary terminology. Two ontology-mediated queries $Q_1 = (\mathbf{S}_1, \Sigma_1, q_1)$ and $Q_2 = (\mathbf{S}_2, \Sigma_2, q_2)$ over a product database, with $\alpha_i = \{j \mid \texttt{Dom}^j \in \mathbf{S}_i\}$, for each $i \in \{1,2\}$, are *comparable relative to schema* $\mathbf{S}$ if $\mathbf{S} = \mathbf{S}_1 \setminus \{\texttt{Dom}^i \mid i \in \alpha_1\} = \mathbf{S}_2 \setminus \{\texttt{Dom}^i \mid i \in \alpha_2\}$. Such comparable queries are *equivalent*, written $Q_1 \equiv Q_2$, if, for every database D over $\mathbf{S}$, $cert_{q_1,\Sigma_1}(D^{\alpha_1}) = cert_{q_2,\Sigma_2}(D^{\alpha_2})$. It is important to say that the above definitions immediately apply even if we consider queries that are not over a product database. An ontology-mediated query language $\mathcal{Q}_2$ is *at least as expressive as* the ontology-mediated query language $\mathcal{Q}_1$, written $\mathcal{Q}_1 \preceq \mathcal{Q}_2$, if, for every query $Q_1 \in \mathcal{Q}_1$ there exists a query $Q_2 \in \mathcal{Q}_2$ such that Q_1 and Q_2 are comparable (relative to some schema) and $Q_1 \equiv Q_2$. $\mathcal{Q}_2$ is *strictly more expressive than* $\mathcal{Q}_1$, written $\mathcal{Q}_1 \prec \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq \mathcal{Q}_2 \not\preceq \mathcal{Q}_1$. Finally, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ *have the same expressive power*, written $\mathcal{Q}_1 = \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq \mathcal{Q}_2 \preceq \mathcal{Q}_1$.

In our analysis we also consider Datalog. It is widely accepted that Datalog is one of the most important database query languages that can be found in the database literature (see, e.g., [10]), and thus it is essential to understand how the ontology-mediated query languages in question compare to it when we focus on product databases. As already said in Section 1, a Datalog program is a set of single-head tgds without existentially quantified variables, while a Datalog query over $\mathbf{S}$ of the form $(\Sigma, \texttt{Ans}/n)$, where Σ is a Datalog program and $\texttt{Ans}$ is the output predicate, can be seen as the ontology-mediated query $(\mathbf{S}, \Sigma, \texttt{Ans}(x_1, \ldots, x_n))$. We write DAT for the class of queries definable via some Datalog query. We show the following:

Theorem 5.1. It holds that,

$$\begin{aligned} (\mathsf{G}, \mathsf{CQ}) \prec (\mathsf{FG}, \mathsf{CQ}) \prec (\mathsf{G}, \mathsf{CQ})_\times = (\mathsf{FG}, \mathsf{CQ})_\times = \mathsf{DAT} \prec \\ (\mathsf{WG}, \mathsf{CQ}) = (\mathsf{WFG}, \mathsf{CQ}) = (\mathsf{WG}, \mathsf{CQ})_\times = (\mathsf{WFG}, \mathsf{CQ})_\times. \end{aligned}$$

The key message of the above result is that the ontology-mediated query languages that are based on (frontier-)guarded existential rules gain in expressiveness when we focus on product databases; in fact, they have the same expressive power as Datalog. However, there is no impact on the expressive power of the ontology-mediated query languages that are based on weakly-(frontier-)guarded existential rules. The rest of this section is devoted to establish the above result. This is done by establishing a series of technical lemmas that all together imply Theorem 5.1. Henceforth, we assume that, given an ontology-mediated query $(\mathbf{S}, \Sigma, q)$, none of the predicates of $\mathbf{S}$ occur in the head of a tgd of Σ. This assumption can be made without loss of generality since, for each $R \in \mathbf{S}$ that appears in the head of tgd of Σ, we can

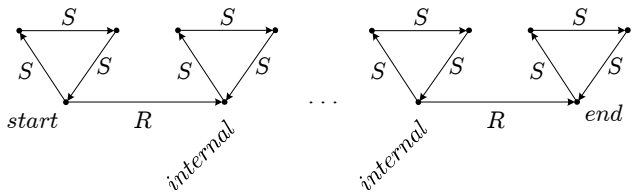

Figure 1. The graph from the proof of Lemma 5.2.

add to Σ the auxiliary copy tgd $R(x_1, \ldots, x_n) \rightarrow R^\star(x_1, \ldots, x_n)$, and then replace each occurrence of R in Σ and q with $R^\star$.

We first establish that frontier-guarded ontology-mediated queries are strictly more expressive than guarded ontology-mediated queries. Although this is generally known, it is not explicitly shown in some previous work. For the sake of completeness, we provide a proof sketch for this fact, leaving the details as an exercise. Our argument is based on the notion of guarded bisimulation, a fundamental tool in the study of guarded logics. In particular, the existence of a guarded bisimulation implies indistinguishability by sentences that fall in guarded fixpoint logic, an extension of the guarded fragment of first-order logic with fixpoint operators; for more details, see [11, 12]. This implies that $(\mathsf{G}, \mathsf{BACQ})$, where BACQ is the class of queries definable by some Boolean atomic CQ, that is, an atomic CQ without free variables, is preserved under guarded bisimulation. In other words, given two $\mathbf{S}$-databases D and D', if they are guarded bisimilar, then $Q(D) \neq \varnothing$ iff $Q(D') \neq \varnothing$, for every $\mathbf{S}$-query $Q \in (\mathsf{G}, \mathsf{BACQ})$.

Lemma 5.2. $(\mathsf{G}, \mathsf{CQ}) \prec (\mathsf{FG}, \mathsf{CQ})$.

Proof:
We need to exhibit a query that can be expressed as a $(\mathsf{FG}, \mathsf{CQ})$ query but not as a $(\mathsf{G}, \mathsf{CQ})$ query, which in turn shows that $(\mathsf{FG}, \mathsf{CQ}) \not\preceq (\mathsf{G}, \mathsf{CQ})$; the other direction holds trivially since $\mathsf{G} \subseteq \mathsf{FG}$. Such a query is the one that asks whether a labeled directed graph $G = (N, E, \lambda, \mu)$, where $\lambda : N \rightarrow \{start, internal, end\}$ and $\mu : E \rightarrow \{R, S\}$, contains a directed R-path P from a start node to an end node via internal nodes, while each node of P is part of a directed S-triangle. In other words, we ask if the graph G contains a subgraph as the one depicted in Figure 1. The graph G is naturally encoded in an $\mathbf{S}$-database D, where $\mathbf{S} = \{\mathtt{Start}/1, \mathtt{Internal}/1, \mathtt{End}/1, R/2, S/2\}$. In fact, D is the database:

$$\begin{aligned} & \{R(v,w) \mid (v,w) \in E \text{ and } \mu(v,w) = R\} \cup \{S(v,w) \mid (v,w) \in E \text{ and } \mu(v,w) = S\} \\ \cup \; & \{\mathtt{Start}(v) \mid v \in N \text{ and } \lambda(v) = start\} \cup \{\mathtt{Internal}(v) \mid v \in N \text{ and } \lambda(v) = internal\} \\ \cup \; & \{\mathtt{End}(v) \mid v \in N \text{ and } \lambda(v) = end\}. \end{aligned}$$

Our query can be expressed as the $(\mathsf{FG}, \mathsf{CQ})$ query $Q = (\mathbf{S}, \Sigma, \mathtt{Yes}())$, where Σ consists of:

$$\begin{aligned} S(x,x_1), S(x_1,x_2), S(x_2,x) &\rightarrow \mathtt{Triangle}_S(x) \\ \mathtt{Start}(x) &\rightarrow \mathtt{Mark}(x) \\ \mathtt{Mark}(x), \mathtt{Triangle}_S(x), R(x,y), \mathtt{Internal}(y) &\rightarrow \mathtt{Mark}(y) \\ \mathtt{Mark}(x), \mathtt{Triangle}_S(x), R(x,y), \mathtt{End}(y), \mathtt{Triangle}_S(y) &\rightarrow \mathtt{Yes}(). \end{aligned}$$

It remains to show that Q cannot be expressed as a $(\mathsf{G}, \mathsf{CQ})$ query. To this end, we first show that Q, and in particular the "triangle checks", cannot be expressed as a $(\mathsf{G}, \mathsf{BACQ})$ query; recall that BACQ

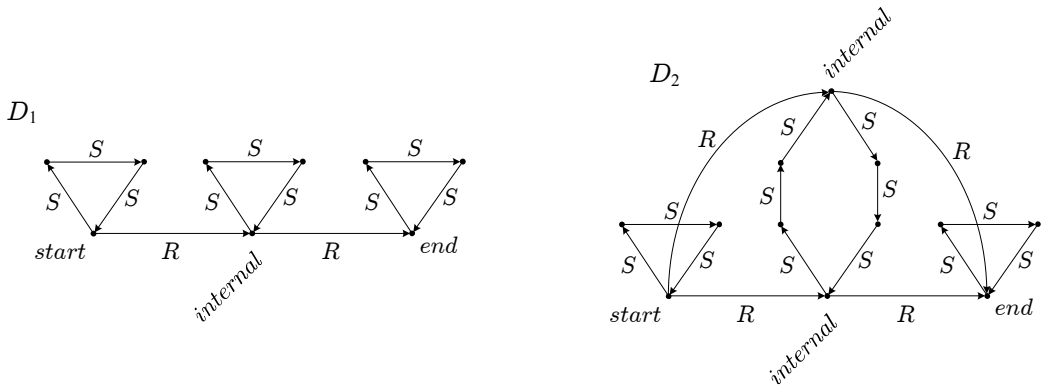

Figure 2. The guarded bisimilar $\mathbf{S}$-databases from the proof of Lemma 5.2.

is the class of queries definable by some Boolean atomic CQ. Towards a contradiction, assume that the query $Q' = (\mathbf{S}, \Sigma', q') \in (\mathsf{G}, \mathsf{BACQ})$ is equivalent to Q. Thus, assuming that D_1 and D_2 are the $\mathbf{S}$-databases depicted in Figure 2, $Q'(D_1) \neq \varnothing$ and $Q'(D_2) = \varnothing$. However, it can be shown that D_1 and D_2 are guarded bisimilar. Since $(\mathsf{G}, \mathsf{BACQ})$ is preserved under guarded bisimulation, we conclude that $Q'(D_1) = \varnothing$ and $Q'(D_2) \neq \varnothing$, which is a contradiction. Consequently, Q cannot be expressed as a $(\mathsf{G}, \mathsf{BACQ})$ query. Now, it is not difficult to argue that Q cannot be expressed as a $(\mathsf{G}, \mathsf{CQ})$ query. Intuitively, the *unbounded* number of "triangle checks" must necessarily be performed by a CQ, which means that we need to express via a CQ an inherently recursive query. But this contradicts the fact that a (finite) first-order query, let alone a conjunctive query, cannot express a recursive query. □

We proceed to show that Datalog queries are strictly more expressive than frontier-guarded ontology-mediated queries. Towards this end, we are going to exploit the fact that $(\mathsf{FG}, \mathsf{ACQ}) \preceq \mathsf{DAT}$, where ACQ is the class of queries definable by some atomic CQ [5].[4] This means that, given a query $Q = (\mathbf{S}, \Sigma, \exists \bar{y}\, \mathtt{Ans}(\bar{x}, \bar{y})) \in (\mathsf{FG}, \mathsf{ACQ})$, there exists a procedure Ξ that translates Σ into a Datalog program such that Q and the query $(\Xi(\Sigma), \mathtt{Ans}) \in \mathsf{DAT}$ over $\mathbf{S}$ are equivalent. We show that:

Lemma 5.3. $(\mathsf{FG}, \mathsf{CQ}) \prec \mathsf{DAT}$.

Proof:
We first show that $(\mathsf{FG}, \mathsf{CQ}) \preceq \mathsf{DAT}$. Let $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{FG}, \mathsf{CQ})$, with $q = \exists \bar{y}\, \phi(x_1, \ldots, x_n, \bar{y})$. Q can be equivalently rewritten as a $(\mathsf{TGD}, \mathsf{ACQ})$ query. More precisely, Q is equivalent to the query

$$Q' \quad = \quad (\mathbf{S}, \Sigma_{P^\star} \cup \Sigma \cup \{\sigma_q\}, \mathtt{Ans}(x_1, \ldots, x_n)),$$

where $\Sigma_{P^\star}$ consists of the tgds:

$$\begin{aligned} R(x_1, \ldots, x_n) &\rightarrow P(x_i), \text{ for each } R \in \mathbf{S} \text{ and } i \in \{1, \ldots, n\} \\ P(x_1), \ldots, P(x_n) &\rightarrow P^\star(x_1, \ldots, x_n), \end{aligned}$$

with $P/1$ and $P^\star/n$ being auxiliary predicates not in $\mathbf{S} \cup sch(\Sigma)$, and σ_q is the tgd

$$P^\star(x_1, \ldots, x_n), \phi(x_1, \ldots, x_n, \bar{y}) \rightarrow \mathtt{Ans}(x_1, \ldots, x_n).$$

[4] A similar result can be found in [13].

In particular, $\Sigma_{P^\star}$ defines the predicate $P^\star$ that holds all the n-tuples over constants of the active domain, which then can be used in σ_q that converts the CQ q into a frontier-guarded tgd. Notice that Q' is not a query over a product database, which means we do not have access to the built-in predicate $\texttt{Dom}^n$. Therefore, in order to convert q into a frontier-guarded tgd, we need to explicitly construct all the n-tuples over the active domain and store them in the auxiliary predicate $P^\star$. Although the set of tgds $\Sigma' = \Sigma_{P^\star} \cup \Sigma \cup \{\sigma_q\}$ is not frontier-guarded, it has a very special form that allows us to rewrite it into a Datalog program by applying the translation Ξ. Observe that Σ' admits a stratification, where the first stratum is the set $\Sigma_{P^\star}$, while the second stratum is the frontier-guarded set $\Sigma \cup \{\sigma_q\}$. This implies that Q' is equivalent to the Datalog query $(\Sigma_{P^\star} \cup \Xi(\Sigma \cup \{\sigma_q\}), \texttt{Ans})$ over $\mathbf{S}$, and the claim follows.

It remains to show that $\mathsf{DAT} \not\preceq (\mathsf{FG}, \mathsf{CQ})$. To this end, it suffices to construct a Datalog query Q over a schema $\mathbf{S}$ such that, for every query $Q' \in (\mathsf{FG}, \mathsf{CQ})$ over $\mathbf{S}$, there exists an $\mathbf{S}$-database D such that, $Q(D) \neq Q'(D)$. We claim that such a Datalog query is Q_{trans} given in Example 4.1, which computes the transitive closure of the binary relation E. Towards a contradiction, assume that Q_{trans} can be expressed as a $(\mathsf{FG}, \mathsf{CQ})$ query $(\mathbf{S}, \Sigma, q)$. Observe that frontier-guarded tgds are not able to put together in an atom, during the construction of the chase instance, two database constants that do not already coexist in a database atom. In particular, given a database D and a set $\Sigma \in \mathsf{FG}$, if there is no atom in D that contains the constants $c, d \in adom(D)$, then there is no atom in $chase(D, \Sigma)$ that contains c and d. Therefore, q is able to compute the transitive closure of the binary relation E. But this contradicts the fact that a (finite) conjunctive query cannot compute the transitive closure of a binary relation. □

We now show that product databases have an impact on the expressiveness of the ontology-mediated query languages based on (frontier-)guarded tgds. In fact, these languages become equally expressive to Datalog when we focus on product databases.

Lemma 5.4. $(\mathsf{G}, \mathsf{CQ})_\times = (\mathsf{FG}, \mathsf{CQ})_\times = \mathsf{DAT}$

Proof:
First observe that $(\mathsf{FG}, \mathsf{CQ})_\times = (\mathsf{FG}, \mathsf{ACQ})_\times$; recall that ACQ is the class of queries definable by some atomic CQ. More precisely, a query $(\mathbf{S}, \Sigma, q) \in (\mathsf{FG}, \mathsf{CQ})_\times$, with $q = \exists \bar{y}\, \phi(x_1, \ldots, x_n, \bar{y})$, is equivalent to the $(\mathsf{FG}, \mathsf{ACQ})_\times$ query

$$(\mathbf{S}, \Sigma \cup \{\sigma_q\}, \texttt{Ans}(x_1, \ldots, x_n)),$$

where σ_q is the tgd

$$\texttt{Dom}^n(x_1, \ldots, x_n), \phi(x_1, \ldots, x_n, \bar{y}) \quad \rightarrow \quad \texttt{Ans}(x_1, \ldots, x_n),$$

which implies that $(\mathsf{FG}, \mathsf{CQ})_\times \preceq (\mathsf{FG}, \mathsf{ACQ})_\times$; the other direction holds trivially. Therefore, to prove our claim, it suffices to show that

$$(\mathsf{G}, \mathsf{CQ})_\times \overset{(1)}{\preceq} (\mathsf{FG}, \mathsf{ACQ})_\times \overset{(2)}{\preceq} \mathsf{DAT} \overset{(3)}{\preceq} (\mathsf{G}, \mathsf{CQ})_\times.$$

For showing (1), we observe that the construction given above for rewriting a $(\mathsf{FG}, \mathsf{CQ})_\times$ query into a $(\mathsf{FG}, \mathsf{ACQ})_\times$ query can be used in order to rewrite a $(\mathsf{G}, \mathsf{CQ})_\times$ query into a $(\mathsf{FG}, \mathsf{ACQ})_\times$ query. For showing (2), we can apply the procedure Ξ mentioned above, which transforms a $(\mathsf{FG}, \mathsf{ACQ})$ query into an equivalent DAT query. Finally, (3) follows from the fact that a Datalog rule ρ can be converted into a guarded tgd by adding in the body of ρ the atom $\texttt{Dom}^{|\bar{x}|}(\bar{x})$, where $\bar{x}$ are the variables in ρ. □

The next lemma shows that weakly-guarded sets of tgds give rise to an ontology-mediated query language that is strictly more expressive than Datalog.

Lemma 5.5. $\mathsf{DAT} \prec (\mathsf{WG}, \mathsf{CQ})$

Proof:
The fact that $\mathsf{DAT} \preceq (\mathsf{WG}, \mathsf{CQ})$ holds trivially since a set of Datalog rules is a weakly-guarded set of tgds. In particular, a Datalog query $(\Sigma, \mathtt{Ans}/n)$ over $\mathbf{S}$ is equivalent to the query $(\mathbf{S}, \Sigma, \mathtt{Ans}(x_1, \ldots, x_n))$, where Σ is trivially weakly-guarded since there are no existentially quantified variables, which in turn implies that the set of affected positions of $sch(\Sigma)$ is empty. It remains to show that $(\mathsf{WG}, \mathsf{CQ}) \not\preceq \mathsf{DAT}$. To this end, we employ a complexity-theoretic argument. It is well-known that the (decision version of the) problem of evaluating a Datalog query is feasible in polynomial time in data complexity, while for $(\mathsf{WG}, \mathsf{CQ})$ is complete for ExpTime [3]. Thus, $(\mathsf{WG}, \mathsf{CQ}) \preceq \mathsf{DAT}$ implies that PTime = ExpTime, which is a contradiction. Therefore, $(\mathsf{WG}, \mathsf{CQ}) \not\preceq \mathsf{DAT}$. □

We finally show that there is no impact on the expressiveness of the ontology-mediated query languages that are based on weakly-(frontier-)guarded sets of tgds. As we shall see, the main reason for this outcome is the fact that these languages are powerful enough to compute the predicates needed to access the active domain.

Lemma 5.6. $(\mathsf{WG}, \mathsf{CQ}) = (\mathsf{WFG}, \mathsf{CQ}) = (\mathsf{WG}, \mathsf{CQ})_\times = (\mathsf{WFG}, \mathsf{CQ})_\times$.

Proof:
It is well-known that $(\mathsf{WG}, \mathsf{CQ}) = (\mathsf{WFG}, \mathsf{CQ})$; $(\mathsf{WG}, \mathsf{CQ}) \preceq (\mathsf{WFG}, \mathsf{CQ})$ holds trivially since $\mathsf{WG} \subseteq \mathsf{WFG}$, while $(\mathsf{WFG}, \mathsf{CQ}) \preceq (\mathsf{WG}, \mathsf{CQ})$ has been shown in [5]. It remains to show $(\mathsf{WG}, \mathsf{CQ}) = (\mathsf{WG}, \mathsf{CQ})_\times$ and $(\mathsf{WFG}, \mathsf{CQ}) = (\mathsf{WFG}, \mathsf{CQ})_\times$.

The $(\preceq)$ direction is trivial. The other direction holds since WG and WFG have the power to explicitly define a predicate P^k/k, where $k > 0$, that holds all the k-tuples of constants in the active domain. More precisely, a $(\mathsf{WG}, \mathsf{CQ})_\times$ (resp., $(\mathsf{WFG}, \mathsf{CQ})_\times$) query $Q = (\mathbf{S}, \Sigma, q)$, with $\alpha = \{j \mid \mathtt{Dom}^j \in \mathbf{S}\}$, is equivalent to the $(\mathsf{WG}, \mathsf{CQ})$ (resp., $(\mathsf{WFG}, \mathsf{CQ})$) query $Q' = (\mathbf{S}', \Sigma', q')$, where $\mathbf{S}' = \mathbf{S} \setminus \{\mathtt{Dom}^k \mid k \in \alpha\}$, Σ' is obtained from Σ by replacing each predicate $\mathtt{Dom}^k$ with P^k and adding the set of tgds:

$$\begin{aligned} R(x_1, \ldots, x_n) &\rightarrow P^1(x_1), \ldots, P^1(x_n), \text{ for each } R \in \mathbf{S}' \\ P^1(x_1), \ldots, P^1(x_k) &\rightarrow P^k(x_1, \ldots, x_k), \text{ for each } k \in \alpha, \end{aligned}$$

and finally q' is obtained from q by replacing each predicate $\mathtt{Dom}^k$ with P^k. □

It is now easy to verify that Lemmas 5.2, 5.3, 5.4, 5.5 and 5.6 imply Theorem 5.1.

6. Is the Existential Quantification Needed?

Theorem 5.1 shows that the query languages based on (frontier-)guarded existential rules over product databases have the same expressive power as plain Datalog. Thus, at this point, one may claim that the existentially quantified variables in the head of a rule are not really needed. However, in applications where we need to decouple the actual database query from the ontology, the existential quantification is

essential; such an application, in the context of querying RDF data, is described in [6]. The goal of this section is to formally show the above intuitive claim. To this end, we adopt a refined notion of expressive power, called *program expressive power*, introduced in [6].

For a fixed set Σ of tgds, we define its program expressive power relative to a schema $\mathbf{S}$, denoted $\mathsf{Pep}_{\mathbf{S}}(\Sigma)$, as the set of triples $(D, q, \bar{c})$, where D is a product $\mathbf{S}$-database, $q(\bar{x})$ is a CQ, $\bar{c} \in \mathbf{C}^{|\bar{x}|}$, and $\bar{c} \in cert_{q,\Sigma}(D)$. In fact, $\mathsf{Pep}_{\mathbf{S}}(\Sigma)$ collects the set of tuples $\bar{c}$ over $\mathbf{C}$ that can be inferred from a product $\mathbf{S}$-database via an OMQ of the form $(\mathbf{S}, \Sigma, \cdot)$. Now, for an ontology-mediated query language $\mathcal{Q} = (\mathcal{C}, \mathsf{CQ})_\times$, where $\mathcal{C} \subseteq \mathsf{TGD}$, we define its program expressive power as the set

$$\mathsf{Pep}(\mathcal{Q}) \;=\; \{\mathsf{Pep}_{\mathbf{S}}(\Sigma) \mid \mathbf{S} \text{ is a schema }, \Sigma \in \mathcal{C}\}.$$

Roughly, $\mathsf{Pep}(\mathcal{Q})$ is a family of sets of triples, where each of its members encodes the program expressive power of a set of tgds in $\mathcal{C}$ relative to some schema. Given two ontology-mediated query languages $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we write $\mathcal{Q}_1 \preceq_{\mathsf{Pep}} \mathcal{Q}_2$ if $\mathsf{Pep}(\mathcal{Q}_1) \subseteq \mathsf{Pep}(\mathcal{Q}_2)$. Finally, we say that $\mathcal{Q}_2$ is *strictly more expressive (w.r.t. the program expressive power)* than $\mathcal{Q}_1$, written $\mathcal{Q}_1 \prec_{\mathsf{Pep}} \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq_{\mathsf{Pep}} \mathcal{Q}_2$ and $\mathcal{Q}_2 \not\preceq_{\mathsf{Pep}} \mathcal{Q}_1$.

We are now ready to illustrate the usefulness of the existentially quantified variables by showing that the ontology-mediated query language based on guarded tgds is more expressive than the language based on existential-free tgds, known in the literature as *full tgds*; we write FULL for the the class of all finite sets of full tgds. Let us clarify that we compare $(\mathsf{G}, \mathsf{CQ})_\times$ with $(\mathsf{FULL}, \mathsf{CQ})_\times$ instead of DAT for compatibility reasons, since the notion of program expressive power is defined for ontology-mediated query languages of the form $(\mathcal{C}, \mathsf{CQ})$, where $\mathcal{C} \subseteq \mathsf{TGD}$. It holds that:

Theorem 6.1. $(\mathsf{FULL}, \mathsf{CQ})_\times \prec_{\mathsf{Pep}} (\mathsf{G}, \mathsf{CQ})_\times$.

Proof:
It is clear that $\mathsf{Pep}((\mathsf{FULL}, \mathsf{CQ})_\times) \subseteq \mathsf{Pep}((\mathsf{G}, \mathsf{CQ})_\times)$ since each $(\mathsf{FULL}, \mathsf{CQ})_\times$ query can be converted into an equivalent $(\mathsf{G}, \mathsf{CQ})_\times$ query by guarding all the variables occurring in the body of a full tgd using a domain predicate of appropriate arity; thus, $(\mathsf{FULL}, \mathsf{CQ})_\times \preceq_{\mathsf{Pep}} (\mathsf{G}, \mathsf{CQ})_\times$. It remains to show that $(\mathsf{G}, \mathsf{CQ})_\times \not\preceq_{\mathsf{Pep}} (\mathsf{FULL}, \mathsf{CQ})_\times$, or, equivalently, $\mathsf{Pep}((\mathsf{G}, \mathsf{CQ})_\times) \not\subseteq \mathsf{Pep}((\mathsf{FULL}, \mathsf{CQ})_\times)$.

Let $\mathbf{S} = \{P\}$, and consider the $\mathbf{S}$-database $D = \{P(c)\}$. Consider also the set Σ consisting of the guarded tgd

$$P(x) \quad \rightarrow \quad \exists y\, R(x, y).$$

Let q_1 and q_2 be the (Boolean) conjunctive queries

$$\exists x \exists y\, R(x, y) \qquad \text{and} \qquad \exists x \exists y (R(x, y) \wedge P(y)),$$

respectively. Clearly, $() \in cert_{q_1,\Sigma}(D)$ and $() \notin cert_{q_2,\Sigma}(D)$, where $()$ denotes the empty tuple. Hence, $(D, q_1, ()) \in \mathsf{Pep}_{\mathbf{S}}(\Sigma)$ and $(D, q_2, ()) \notin \mathsf{Pep}_{\mathbf{S}}(\Sigma)$, which in turn implies that $\mathsf{Pep}((\mathsf{G}, \mathsf{CQ})_\times)$ contains a set of triples T such that $(D, q_1, ()) \in T$ and $(D, q_2, ()) \notin T$. We claim that $T \notin \mathsf{Pep}((\mathsf{FULL}, \mathsf{CQ})_\times)$, and thus, $\mathsf{Pep}((\mathsf{G}, \mathsf{CQ})_\times) \not\subseteq \mathsf{Pep}((\mathsf{FULL}, \mathsf{CQ})_\times)$, as needed.

It is not difficult to see that *for every* set Σ' of full tgds, $() \in cert_{q_1,\Sigma'}(D)$ implies $() \in cert_{q_2,\Sigma'}(D)$. Thus, the triples $(D, q_1, ())$ and $(D, q_2, ())$ necessarily coexist in $\mathsf{Pep}_{\mathbf{S}}(\Sigma')$, for every set $\Sigma' \in \mathsf{FULL}$, which in turn implies that $T \notin \mathsf{Pep}((\mathsf{FULL}, \mathsf{CQ})_\times)$, and the claim follows. □

Class $\mathcal{C}$	Data Complexity	Bounded Arity	Combined Complexity
G	PTIME	EXPTIME	2EXPTIME
FG	PTIME	2EXPTIME	2EXPTIME
WG	EXPTIME	EXPTIME	2EXPTIME
WFG	EXPTIME	2EXPTIME	2EXPTIME

Table 1. Complexity of $\text{EVAL}((\mathcal{C}, \mathsf{CQ})_\times)$; all the results are completeness results.

7. Complexity of Query Evaluation

The question that remains to be answered is whether product databases have an impact on the complexity of the query evaluation problem under the guarded-based ontology-mediated query languages in question. As is customary when studying the computational complexity of the evaluation problem for a query language, we consider its associated decision problem. We denote this problem by $\text{EVAL}(\mathcal{Q})$, where $\mathcal{Q}$ is an ontology-mediated query language, and its definition follows:

INPUT :	Query $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in \mathcal{Q}$, $\mathbf{S}$-database D, and tuple $\bar{t} \in \mathbf{C}^{\lvert\bar{x}\rvert}$.
QUESTION :	Does $\bar{t} \in cert_{q,\Sigma}(D)$?

It is important to clarify that when we focus on ontology-mediated queries over a product database, then the input database to the evaluation problem is a product database. In other words, if we focus on the problem $\text{EVAL}((\mathcal{C}, \mathsf{CQ})_\times)$, where $\mathcal{C}$ is a class of tgds, and the input query is $(\mathbf{S}, \Sigma, q)$, then the input database is an α-product database, where $\alpha = \{i \mid \mathtt{Dom}^i \in \mathbf{S}\}$.

The complexity of $\text{EVAL}((\mathcal{C}, \mathsf{CQ}))$, where $\mathcal{C} \in \{\mathsf{G}, \mathsf{FG}, \mathsf{WG}, \mathsf{WFG}\}$, is well-understood; for $(\mathsf{G}, \mathsf{CQ})$ and $(\mathsf{WG}, \mathsf{CQ})$ it has been investigated in [3], while for $(\mathsf{FG}, \mathsf{CQ})$ and $(\mathsf{WFG}, \mathsf{CQ})$ in [14]. It is clear that the algorithms devised in [3, 14] for the guarded-based ontology-mediated query languages in question treat product databases in the same way as non-product databases, or, in other words, they are oblivious to the fact that an input database is product. Therefore, we can conclude that, even if we focus on product databases, the existing algorithms can be applied and get the same complexity results for query evaluation as in the case where we consider arbitrary (non-product) databases; these results are summarized in Table 1. Recall that the data complexity is calculated by considering only the database as part of the input, while in the combined complexity both the query and the database are part of the input. We also consider the important case where the arity of the schema is bounded by an integer constant.

7.1. The Bounded Arity Case Revisited

In Table 1, the bounded arity column refers to the case where all predicates in the given query, including the predicates of the form $\mathtt{Dom}^k$, where $k > 0$, are of bounded arity. However, bounding the arity of the $\mathtt{Dom}^k$ predicates is not our intention. Observe that in the proof of Lemma 5.4, where we show $(\mathsf{G}, \mathsf{CQ})_\times = (\mathsf{FG}, \mathsf{CQ})_\times = \mathsf{DAT}$, the predicates of the form $\mathtt{Dom}^k$ are used (i) to convert a CQ into a frontier-guarded tgd, and (ii) to convert a Datalog rule into a guarded tgd. More precisely, in the first case we use a $\mathtt{Dom}^k$ atom to guard the answer variables of a CQ, while in the second case to guard the variables in the body of a Datalog rule. Therefore, in both cases, we need to guard via a $\mathtt{Dom}^k$ atom

an unbounded number of variables, even if the arity of the schema is bounded, and thus k must be unbounded. From the above discussion, it is clear that the interesting case to consider in our complexity analysis is not when all predicates of the underlying schema are of bounded arity, but when all predicates except the domain predicates are of bounded arity. Clearly, in case of $(\mathsf{FG}, \mathsf{CQ})_\times$ and $(\mathsf{WFG}, \mathsf{CQ})_\times$, the complexity of query evaluation is 2EXPTIME-complete, since the problem is 2EXPTIME-hard even if all predicates (including the domain predicates) have bounded arity. However, the picture is foggy in the case of $(\mathsf{G}, \mathsf{CQ})_\times$ and $(\mathsf{WG}, \mathsf{CQ})_\times$ since the existing results imply a 2EXPTIME upper bound and an EXPTIME lower bound. Interestingly, as we discuss below, the complexity of query evaluation remains the same, i.e., EXPTIME-complete, even if the domain predicates have unbounded arity.

Theorem 7.1. $\text{EVAL}((\mathsf{WG}, \mathsf{CQ})_\times)$ is EXPTIME-complete if the arity of the schema, excluding the predicates of the form $\mathtt{Dom}^k$, where $k > 0$, is bounded by an integer constant.

The lower bound follows from the fact $\text{EVAL}((\mathsf{WG}, \mathsf{CQ}))$ is EXPTIME-hard when the arity of the schema is bounded [3]. The upper bound relies on a result that, although being implicit in [3], it has not been explicitly stated before. The *body-predicates* of an ontology-mediated query $(\mathbf{S}, \Sigma, q)$ are the predicates that do not appear in the head of a tgd of Σ. It holds that:

Proposition 7.2. $\text{EVAL}((\mathsf{WG}, \mathsf{CQ})_\times)$ is in EXPTIME if the arity of the schema, excluding the body-predicates, is bounded by an integer constant.

The above result simply states that even if we allow the body-predicates to have unbounded arity, while all the other predicates of the schema are of bounded arity, the complexity of $\text{EVAL}((\mathsf{WG}, \mathsf{CQ})_\times)$ remains the same as in the case where all the predicates of the schema have bounded arity. Since the predicates of the form $\mathtt{Dom}^k$, for $k > 0$, is a subset of the body-predicates of an ontology-mediated query over a product database, it is clear that Proposition 7.2 implies Theorem 7.1.

As said, although Proposition 7.2 has not been explicitly stated before, it is implicit in [3], where the complexity of query evaluation for $(\mathsf{WG}, \mathsf{CQ})$ is investigated. In fact, we can apply the alternating algorithm devised in [3] for showing that $\text{EVAL}((\mathsf{WG}, \mathsf{CQ}))$ is in EXPTIME if the arity of the schema (including the body-predicates) is bounded by an integer constant. In what follows, we briefly recall the main ingredients of the alternating algorithm in [3], and discuss how we get the desired upper bound.

A set $\Sigma \in \mathsf{WG}$ can be effectively transformed into a set $\Sigma' \in \mathsf{WG}$ such that all the tgds of Σ' are single-head [3]. Henceforth, for technical clarity, we focus on tgds with just one atom in the head. Let D be a database, and Σ a set of tgds. Fix a chase sequence $D = I_0 \xrightarrow{\sigma_0, h_0} I_1 \xrightarrow{\sigma_1, h_1} I_2 \ldots$ for D under Σ. The instance $chase(D, \Sigma)$ can be naturally represented as a labeled directed graph $G = (N, E, \lambda)$ as follows: (1) for each atom $R(\bar{t}) \in chase(D, \Sigma)$, there exists $v \in N$ such that $\lambda(v) = R(\bar{t})$; (2) for each $i \geq 0$, with $I_i \xrightarrow{\sigma_i, h_i} I_{i+1}$, and for each atom $R(\bar{t}) \in h_i(body(\sigma_i))$, there exists $(v, u) \in E$ such that $\lambda(v) = R(\bar{t})$ and $\{\lambda(u)\} = I_{i+1} \setminus I_i$; and (3) there are no other nodes and edges in G. The *guarded chase forest* of D and Σ, denoted $\mathsf{gcf}(D, \Sigma)$, is the forest obtained from G by keeping only the nodes associated with weak-guards, and their children; for more details, we refer the reader to [3].

Consider a query $(\mathbf{S}, \Sigma, q) \in (\mathsf{WG}, \mathsf{CQ})$, a database D over $\mathbf{S}$, and a tuple $\bar{t}$ of constants. Clearly, $\bar{t} \in cert_{q,\Sigma}(D)$ iff there exists a homomorphism that maps $q(\bar{t})$ to $\mathsf{gcf}(D, \Sigma)$. Observe that if such a homomorphism h exists, then in $\mathsf{gcf}(D, \Sigma)$ there exist paths starting from nodes labeled with database atoms and ending at nodes labeled with the atoms of $h(q(\bar{t}))$. The alternating algorithm in [3] first guesses

the homomorphism h from $q(\bar{t})$ to $\mathsf{gcf}(D, \Sigma)$, and then constructs in parallel universal computations the paths from D to $h(q(\bar{t}))$ (if they exist). During this alternating process, the algorithm exploits a key result established in [3], that is, the subtree of $\mathsf{gcf}(D, \Sigma)$ rooted at some atom $R(\bar{u})$ is determined by the so-called cloud of $R(\bar{u})$ (modulo renaming of nulls) [3, Theorem 5.16]. The *cloud* of $R(\bar{u})$ with respect to D and Σ, denoted $cloud(R(\bar{u}), D, \Sigma)$, is defined as $\{S(\bar{v}) \in chase(D, \Sigma) \mid \bar{v} \subseteq (adom(D) \cup \bar{u})\}$, i.e., the atoms in the result of the chase with constants from D and terms from $\bar{u}$. This result allows the algorithm to build the relevant paths of $\mathsf{gcf}(D, \Sigma)$ from D to $h(q(\bar{t}))$. Roughly, an atom $R(\bar{u})$ on a path can be generated by considering only its parent atom $S(\bar{v})$ and the cloud of $S(\bar{v})$ with respect to D and Σ. Whenever a new atom is generated, the algorithm nondeterministically guesses its cloud, and verify in a parallel universal computation that indeed belongs to the result of the chase.

From the above informal description, we conclude that the space needed at each step of the computation of the alternating algorithm is actually the size of the cloud of an atom. By applying a simple combinatorial argument, it is easy to show that the size of a cloud is at most $(|\mathbf{S}|+|sch(\Sigma)|)\cdot(|adom(D)|+w)^w$, where w is the maximum arity over all predicates of $\mathbf{S} \cup sch(\Sigma)$. Therefore, if we assume that all the predicates of the schema have bounded arity, which means that w is a constant, then the size of a cloud is polynomial. Since alternating polynomial space coincides with deterministic exponential time, we immediately get the ExpTime upper bound in the case of bounded arity. Now, let $\mathbf{B}$ be the body-predicates of $(\mathbf{S}, \Sigma, q)$. It is clear that, for every atom $R(\bar{u}) \in chase(D, \Sigma)$, the restriction of $cloud(R(\bar{u}), D, \Sigma)$ on the predicates of $\mathbf{B}$ is actually D, and thus of polynomial size, even if the predicates of $\mathbf{B}$ have unbounded arity. This implies that, even if we allow body-predicates of unbounded arity, the size of a cloud remains polynomial. Therefore, the alternating algorithm devised in [3] can be applied in order to get the ExpTime upper bound stated in Proposition 7.2.

Body-Predicates vs. Data Schema

As already discussed in Section 5 (see the paragraph after Theorem 5.1), we can assume that, given an ontology-mediated query $(\mathbf{S}, \Sigma, q)$, none of the predicates of $\mathbf{S}$ occur in the head of a tgd of Σ. Therefore, according to this assumption, the predicates of the data schema are body-predicates. In view of this fact, one may claim that Proposition 7.2 holds even if we allow the data schema to have unbounded arity. However, this is not true since in this case already $\text{Eval}((\mathsf{G}, \mathsf{CQ}))$ is 2ExpTime-hard. Since the predicates of the data schema may appear in the head of a tgd, it is straightforward to reduce $\text{Eval}((\mathsf{G}, \mathsf{CQ}))$ where all the predicates of the schema have unbounded arity to $\text{Eval}((\mathsf{G}, \mathsf{CQ}))$ where only the data schema has unbounded arity; given an arbitrary query $(\mathbf{S}, \Sigma, q) \in (\mathsf{G}, \mathsf{CQ})$ simply take the query $(\mathbf{S}', \Sigma, q)$, where $\mathbf{S}' = \mathbf{S} \cup sch(\Sigma)$, i.e., all the predicates are part of the data schema, and thus can have unbounded arity.

The reason why we assume that none of the predicates of $\mathbf{S}$ occur in the head of a tgd of Σ is because we add in Σ the tgd $R(x_1, \ldots, x_n) \rightarrow R^\star(x_1, \ldots, x_n)$, for each $R \in \mathbf{S}$ that appears in the head of a tgd, and then replace each occurrence of the predicate R in Σ and q with $R^\star$. Hence, if we allow the predicates of the data schema to have unbounded arity, then, after adding the auxiliary copy rules in Σ and renaming the relevant predicates R to $R^\star$, several predicates of unbounded arity, which are not body-predicates, occur in Σ. This immediately increases the size of the cloud of an atom, which becomes exponential. Since alternating exponential space coincides with deterministic double-exponential time, the alternating algorithm described above provides a 2ExpTime upper bound for $\text{Eval}((\mathsf{WG}, \mathsf{CQ})_\times)$ when the data schema has unbounded arity, while all the other predicates are of bounded arity.

A Note on Succinctness

The next question that comes up concerns the succinctness of the obtained query languages after considering product databases. This challenging problem goes beyond the scope of this work, and is something that we are currently investigating. Nevertheless, we would like to present a result, which can be seen as a strong indication that frontier-guardedness allows us to build more succinct queries. More precisely, an interesting consequence of Theorem 7.1 is that, although $(\mathsf{FG}, \mathsf{CQ})_\times$ and $(\mathsf{G}, \mathsf{CQ})_\times$ are equally expressive, there is no a polynomial time arity-preserving translation from the former to the latter; the same holds for the weak versions of those languages. It is easy to show that:

Proposition 7.3. There exists $Q \in (\mathsf{FG}, \mathsf{CQ})_\times$ (resp., $(\mathsf{WFG}, \mathsf{CQ})_\times$) that is not equivalent to any query $Q' \in (\mathsf{G}, \mathsf{CQ})_\times$ (resp., $(\mathsf{WG}, \mathsf{CQ})_\times$) that can be constructed in polynomial time.

Proof:
Towards a contradiction, assume that for every $Q \in (\mathsf{FG}, \mathsf{CQ})_\times$ (resp., $(\mathsf{WFG}, \mathsf{CQ})_\times$) we can construct in polynomial time a query $Q' \in (\mathsf{G}, \mathsf{CQ})_\times$ (resp., $(\mathsf{WG}, \mathsf{CQ})_\times$) that is equivalent to Q. This implies that we can reduce in polynomial time a 2EXPTIME-hard problem into a problem that is feasible in EXPTIME (the latter holds by Theorem 7.1). Therefore, EXPTIME coincides with 2EXPTIME, which is a contradiction since EXPTIME $\subsetneq$ 2EXPTIME. □

8. The Language Perspective

As discussed above (see the proof of Lemma 5.3), one of the crucial limitations of (frontier-)guarded tgds is the fact that are not powerful enough to compute the transitive closure of a binary relation. Actually, the technical reason for this is the fact that (frontier-)guarded tgds are not able to put together in an atom, during the construction of the chase instance, two database constants that do not already coexist in a database atom. This has recently motivated the definition of two refined classes of tgds, called *nearly (frontier-)guarded*, which allow for non(-frontier)-guarded rules as long as, for each body-variable V, at least one occurrence of V occurs at a non-affected position [5]. Formally, a set Σ of tgds is nearly (frontier-)guarded if, for each $\sigma \in \Sigma$, the following holds: (i) σ is (frontier-)guarded or, (ii) for each variable V occurring in $body(\sigma)$, at least once occurrence of V appears at a position $\pi \notin affected(\Sigma)$. Let NG (resp., NFG) be the class of all finite nearly guarded (resp., nearly frontier-guarded) sets of tgds. By exploiting the expressive power analysis performed in [5], we can easily show that the ontology-mediated query languages based on nearly (frontier-)guarded sets of tgds are equally expressive to Datalog. This fact, together with Lemmas 5.2 and 5.3, immediately implies that

$$(\mathsf{G}, \mathsf{CQ}) \prec (\mathsf{FG}, \mathsf{CQ}) \prec (\mathsf{NG}, \mathsf{CQ}) = (\mathsf{NFG}, \mathsf{CQ}) = \mathsf{DAT}.$$

It is also important to say the evaluation problem for both $(\mathsf{NG}, \mathsf{CQ})$ and $(\mathsf{NFG}, \mathsf{CQ})$ remains tractable in data complexity; implicit in [5].

From the above discussion, it is clear that $(\mathsf{NG}, \mathsf{CQ})$ and $(\mathsf{NFG}, \mathsf{CQ})$ are central ontology-mediated query languages, which can be used in the place of Datalog without losing in expressive power. In fact, $(\mathsf{NG}, \mathsf{CQ})$ and $(\mathsf{NFG}, \mathsf{CQ})$ allow us to write more intuitive and compact queries than Datalog due to the existential quantification in rule-heads; recall the discussion on program expressive power in Section 6. Nevertheless, by adopting $(\mathsf{NG}, \mathsf{CQ})$ and $(\mathsf{NFG}, \mathsf{CQ})$, we lose one of the key advantages of Datalog, that

is, the fact that we can write queries in a modular way. More precisely, given two Datalog programs Σ_1 and Σ_2, $\Sigma_1 \cup \Sigma_2$ is also a Datalog program, i.e., Datalog is closed under union, which is a key property for a rule-based query language that enables a modular style of writing queries.[5] Unfortunately, as illustrated by the following example, this not true for nearly (frontier-)guardedness since its definition relies on the (global) notion of affected position, which takes into account the whole set of tgds.

Example 8.1. Consider the two sets of tgds:

$$\Sigma_1 \;=\; \{P(x,y), P(y,z) \rightarrow P(x,z)\} \qquad \Sigma_2 \;=\; \{P(x,y) \rightarrow \exists z\, P(y,z)\}.$$

It is straightforward to verify that both Σ_1 and Σ_2 are nearly guarded; in fact, Σ_2 is guarded. However, $\Sigma_1 \cup \Sigma_2$ is not nearly-guarded. Actually, $\Sigma_1 \cup \Sigma_2$ is not even weakly-frontier-guarded. The reason for this is because $P[1], P[2] \in \mathit{affected}(\Sigma_1 \cup \Sigma_2)$. ∎

The question that comes up is whether we can define classes of tgds that: (i) syntactically generalize (frontier-)guarded tgds, (ii) are closed under union, and (iii) give rise to data tractable ontology-mediated query languages that are equally expressive to Datalog. Interestingly, we can exploit the results of Section 5 in order to define such classes of tgds, which are actually based on the notion of marked tgds.

Marked Tgds and Ontology-Mediated Queries. A *marked tgd* is a pair (σ, M), where σ is a tgd, and M is a set of variables, called *marked variables*, occurring in the body of σ. The idea is to force the marked variables to be satisfied by constants and not nulls. Formally, an instance I *satisfies* the marked tgd $(\phi(\bar{x},\bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x},\bar{z}), \{v_1,\ldots,v_n\})$, where $\{v_1,\ldots,v_n\} \subseteq \bar{x} \cup \bar{y}$, if: For every homomorphism h from $\phi(\bar{x},\bar{y})$ to I such that $\{h(v_1),\ldots,h(v_n)\} \subset \mathbf{C}$, there is a homomorphism h' that extends h, i.e., $h' \supseteq h$, from $\psi(\bar{x},\bar{z})$ to I. I satisfies a set Σ of marked tgds, denoted $I \models_{\mathbf{C}} \Sigma$, if I satisfies every marked tgd in Σ. Notice that the only difference between the classical notion of entailment ($\models$) and our new notion of entailment ($\models_{\mathbf{C}}$) is the fact the body of a marked tgd is satisfied by an instance I, via a homomorphism h, only if the marked variables are mapped via h to constants occurring in I.

It is now quite natural to consider ontology-mediated queries based on marked tgds. Let $(\mathbf{S}, \Sigma, q)$ be an ontology-mediated query, where Σ is a set of marked tgds, and n is the arity of q. The *answer* to q with respect to a database D over $\mathbf{S}$ and Σ is the set of tuples

$$\mathit{cert}^{\mathbf{C}}_{q,\Sigma}(D) \;=\; \bigcap_{I \supseteq D, I \models_{\mathbf{C}} \Sigma} \{\bar{c} \in \mathit{adom}(D)^n \mid \bar{c} \in q(I)\}.$$

We use the superscript $\mathbf{C}$ in order to make explicit that we employ our new notion of entailment ($\models_{\mathbf{C}}$). Let us clarify that in case of plain tgds, which are actually marked tgds with an empty set of marked variables, $\mathit{cert}_{q,\Sigma}(D)$ and $\mathit{cert}^{\mathbf{C}}_{q,\Sigma}(D)$ coincide.

Recall that for an ontology-mediated query $(\mathbf{S}, \Sigma, q)$, where Σ is a set of (plain) tgds, $\mathit{cert}_{q,\Sigma}(D) = q(\mathit{chase}(D,\Sigma))$, for every $\mathbf{S}$-database D. An analogous result can be shown for ontology-mediated queries that are based on marked tgds. To this end, we employ a slightly modified version of the chase procedure, which is obtained by adapting the notion of applicability of a tgd. Let I be an instance and $\sigma = (\phi(\bar{x},\bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x},\bar{z}), \{v_1,\ldots,v_n\})$ a marked tgd. We say that σ is *applicable* with respect to I if there exists a homomorphism h from $\phi(\bar{x},\bar{y})$ to I such that $h(v_i) \in \mathbf{C}$, for each $i \in \{1,\ldots,n\}$.

[5] This is also true for (frontier-)guarded tgds.

By employing this notion of applicability, we can naturally define the chase for an instance I under a set Σ of marked tgds, denoted $\mathbf{C}\text{-}chase(I, \Sigma)$. It is then not difficult to show the following: Given an ontology-mediated query $(\mathbf{S}, \Sigma, q)$, where Σ is a set of marked tgds, $cert^{\mathbf{C}}_{q,\Sigma}(D) = q(\mathbf{C}\text{-}chase(D, \Sigma))$, for every $\mathbf{S}$-database D.

At this point, we would like to stress that marked tgds are similar in spirit to *DL-safe rules* introduced in [15]. The goal of [15] was to combine description logics (DLs) and Datalog in such a way that the decidability of key reasoning tasks, e.g., satisfiability and query answering, is preserved. This is achieved by forcing the Datalog rules to be DL-safe, which means that each variable x in a rule is required to occur in a non-DL-atom, or, in other words, at least one occurrence of x should appear in an atom of the form $R(\bar{t})$, where R is a predicate that is not mentioned by the DL axioms. The fact that a Datalog rule is DL-safe essentially says that it suffices to instantiate the variables occurring in the rule with existing database constants. Consequently, both DL-safe rules and marked tgds provide a mechanism for ensuring that the variables occurring in a rule can be safely instantiated with existing database constants.

(Frontier-)Guarded Marked Tgds. We proceed to defined (frontier-)guarded marked tgds, which are actually the formalisms that we are looking for. A marked tgd (σ, M) is *guarded* (resp., *frontier-guarded*) if the following holds: (i) σ is guarded (resp., frontier-guarded), or (ii) M is the set of variables occurring in the body (resp., frontier) of σ. Let GM (resp., FGM) the classes of all finite sets of guarded (resp., frontier-guarded) marked tgds. Observe that a (frontier-)guarded tgd σ can be straightforwardly rewritten as the (frontier-)guarded marked tgd $(\sigma, \varnothing)$. Moreover, both GM and FGM are, by definition, closed under union. Finally, it is an easy exercise to show that the ontology-mediated query languages $(\mathsf{GM}, \mathsf{CQ})$ and $(\mathsf{FGM}, \mathsf{CQ})$ are data tractable and equally expressive to Datalog. The idea is to rewrite a query in $(\mathsf{G}, \mathsf{CQ})_\times$ (resp., $(\mathsf{FG}, \mathsf{CQ})_\times$) into a query $(\mathsf{GM}, \mathsf{CQ})$ (resp., $(\mathsf{FGM}, \mathsf{CQ})$) by converting the atoms of the form $\mathtt{Dom}^{\mathtt{n}}(x_1, \ldots, x_n)$ into sets of marked variables $\{x_1, \ldots, x_n\}$ and vice versa.

Theorem 8.2. The following holds:

1. $\textsc{Eval}((\mathcal{C}, \mathsf{CQ}))$, where $\mathcal{C} \in \{\mathsf{GM}, \mathsf{FGM}\}$, is PTIME-complete in data complexity.
2. $(\mathsf{GM}, \mathsf{CQ}) = (\mathsf{FGM}, \mathsf{CQ}) = \mathsf{DAT}$.

9. Conclusions

It is realistic to assume that a database management system provides access to the active domain via built-in relations, or, in more formal terms, to assume that queries are evaluated over product databases. Interestingly, the query languages that are based on (frontier-)guarded existential rules gain in expressiveness when we focus on product databases; in fact, they have the same expressive power as Datalog. On the other hand, there is no impact on the expressive power of the query languages based on weakly-(frontier-)guarded existential rules, since they are powerful enough to explicitly compute the predicates needed to access the active domain. We also observe that there is no impact on the computational complexity of the query languages in question. Finally, our expressive power results under the assumption of product databases, helped to define novel classes of tgds that give rise to query languages that are equally expressive to Datalog, data tractable, and closed under union. Recall that the latter property is crucial for a query language, which allows us to write queries in a modular way.

Acknowledgements: Gottlob and Pieris are supported by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems Principles and Architecture". Šimkus is supported by the Austrian Science Fund (FWF), projects P25207-N23 and Y698.

References

[1] Bienvenu M, ten Cate B, Lutz C, Wolter F. Ontology-Based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP. ACM Trans Database Syst. 2014;39(4):33:1–33:44.

[2] Beeri C, Vardi MY. The Implication Problem for Data Dependencies. In: ICALP; 1981. p. 73–85.

[3] Calì A, Gottlob G, Kifer M. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. J Artif Intell Res. 2013;48:115–174.

[4] Baget JF, Leclère M, Mugnier ML, Salvat E. On rules with existential variables: Walking the decidability line. Artif Intell. 2011;175(9-10):1620–1654.

[5] Gottlob G, Rudolph S, Simkus M. Expressiveness of guarded existential rule languages. In: PODS; 2014. p. 27–38.

[6] Arenas M, Gottlob G, Pieris A. Expressive languages for querying the semantic web. In: PODS; 2014. p. 14–26.

[7] Fagin R, Kolaitis PG, Miller RJ, Popa L. Data exchange: Semantics and query answering. Theor Comput Sci. 2005;336(1):89–124.

[8] Johnson DS, Klug AC. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. J Comput Syst Sci. 1984;28(1):167–189.

[9] Maier D, Mendelzon AO, Sagiv Y. Testing Implications of Data Dependencies. ACM Trans Database Syst. 1979;4(4):455–469.

[10] Abiteboul S, Hull R, Vianu V. Foundations of Databases. Addison-Wesley; 1995.

[11] Andréka H, van Benthem J, Németi I. Modal Languages and Bounded Fragments of Predicate Logic. J Philosophical Logic. 1998;27:217–274.

[12] Grädel E. Decision procedures for guarded logics. In: CADE; 1999. p. 31–51.

[13] Bárány V, Benedikt M, ten Cate B. Rewriting Guarded Negation Queries. In: MFCS; 2013. p. 98–110.

[14] Baget JF, Mugnier ML, Rudolph S, Thomazo M. Walking the Complexity Lines for Generalized Guarded Existential Rules. In: IJCAI; 2011. p. 712–717.

[15] Motik B, Sattler U, Studer R. Query Answering for OWL-DL with rules. J Web Sem. 2005;3(1):41–60.